Freecell/Solitaire overview

- What are the use cases?
  - How does customer use the program?
  - What are scenarios as the game develops?
  - What parts of the “standard version” are good/bad?
  - What options might we want to have?

- How will we design the program?
  - Brainstorm classes
  - Develop and test
  - Rethink design and use cases
  - Develop and test
  - ...

Freecell classes

- What are the classes in the program? Behaviors?
  - Look for objects, how do they act? Nouns? Verbs

- What about a Card class? Behaviors/Responsibilities?
  - State? Mutable?
  - Comparison? Other games?
  - Who creates cards? Relevant to solitaire? Other games?

- What about CardPile classes, similarities? Differences?
  - FreeCell, AcePile, DrawPile, ...
  - Other card games?

Inheritance (language independent)

- First view: exploit common interfaces in programming
  - iterator, C++ function objects
    - Iterators in STL/C++ share interface by convention/templates
    - Implementation varies while interface stays the same

- Second view: share code, factor code into parent class
  - Code in parent class shared by subclasses
  - Subclasses can override inherited method
    - Can subclasses override and call?

- Polymorphism/late(runtime) binding (compare: static)
  - Actual function called determined when program runs, not when program is compiled

Inheritance guidelines in C++

- Inherit from Abstract Base Classes (ABC)
  - one pure virtual function needed (=0)
    - Subclasses must implement, or they’re abstract too
  - must have virtual destructor implemented
    - can have pure virtual destructor with an implementation, but this is special case, not normally needed [force ABC]

- Avoid protected data, but sometimes this isn’t possible
  - data is private, subclasses have it, can’t access it
  - keep protected data to a minimum

- Single inheritance, assume most functions are virtual
  - multiple inheritance ok when using ABC, problem with data in super classes
  - virtual: some overhead, but open/closed principle intact
### Inheritance Heuristics

- A base/parent class is an interface
  - Subclasses implement the interface
    - Behavior changes in subclasses, but there’s commonality
  - The base/parent class can supply some default behavior
    - Derived classes can use, override, both
  - The base/parent class can have state
    - Protected: inherited and directly accessible
    - Private: inherited but not accessible directly
  - Abstract base classes are a good thing
- Push common behavior high up in an inheritance hierarchy
- If the subclasses aren’t used polymorphically (e.g., through a pointer to the base class) then the inheritance hierarchy is probably flawed

### Inheritance Heuristics in C++

- One pure virtual (aka abstract) function makes a class abstract
  - Cannot be instantiated, but can be constructed (why?)
  - Default in C++ is non-virtual or monomorphic
    - Unreasonable emphasis on efficiency, sacrifices generality
    - If you think subclassing will occur, all methods are virtual
  - Must have virtual destructor, the base class destructor (and constructor) will be called
- We use public inheritance, models is-a relationship
  - Private inheritance means is-implemented-in-terms-of
    - Implementation technique, not design technique
    - Derived class methods call base-class methods, but no “usable-as-a” via polymorphism
    - Access to protected methods, and can redefine virtual_funcs

### Inheritance and Layering/Aggregation

- Layering (or aggregation) means “uses via instance variable”
  - Use layering/attributes if differences aren’t behavioral
  - Use inheritance when differences are behavioral
- Consider Student class: name, age, gender, sleeping habits
  - Which are attributes, which might be virtual methods
- Lots of classes can lead to lots of problems
  - It’s hard to manage lots of classes in your head
  - Tools help, use speedbar in emacs, other class browsers in IDEs or in comments (e.g., javadoc)
- Inheritance hierarchies cannot be too deep (understandable?)

### Inheritance guidelines (from Riel)

- Beware derived classes with only one instance/object
  - For the CarMaker class is GeneralMotors a subclass or an object?
- Beware derived classes that override behavior with a no-op
  - Mammal class from which platypus derives, live-birth?
- Too much subclassing? Base class House
  - Derived: ElectricallyCooledHouse, SolarHeatedHouse?
- What to do with a list of fruit that must support apple-coring?
  - Fruit list is polymorphic (in theory), not everything corable